REPORT DOCUMENTATION PAGE

AFRL-SR-BL-TR-00-

0632

1. REPORT DATE (DD-MM-YYYY)	2. REPORT DATE	3. DATES COVERED (From - To)
25-09-2000	Final Technical	1-Jun-96 to 30-May-00

4. TITLE AND SUBTITLE

Stochastic Scheduling and Planning Using Reinforcement Learning

5a. CONTRACT NUMBER

5b. GRANT NUMBER

F49620-96-1-0254

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Barto, Andrew G
Computer Science Department
140 Governors Drive
University of Massachusetts
Amherst, MA 01003

5d. PROJECT NUMBER

2304/ AX

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

University of Massachusetts
Computer Science Department
Amherst, MA 01003

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Office of Scientific Research
801 N Randolph St Room 732
Arlington, VA 22203-1977

10. SPONSOR/MONITOR'S ACRONYM(S)

AFOSR/NM

11. SPONSORING/MONITORING AGENCY REPORT NUMBER

12. DISTRIBUTION AVAILABILITY STATEMENT

Approved for public release,
distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT This project investigated the extension of reinforcement learning (RL) methods to large-scale optimization problems relevant to Air Force operations planning, scheduling, and maintenance. The objectives of this project were to: 1) investigate the utility of RL on large-scale logistics problems; 2) extend existing RL theory and practice to enhance the ease of application and the performance of RL on these problems; and 3) explore new problem formulations in order to take maximal advantage of RL methods. A method using RL to modify local search cost functions was developed and shown to yield significant improvement over a traditional local search method on a core vehicle routing problem. A new method for stochastic dynamic optimization was studied, a theoretical result proven, and utility demonstrated using a simulated aerial mission planning task. A learning-based method for optimizing subproblem selection in divide-and-conquer approaches was developed and demonstrated on graph-coloring and on a multiple-vehicle mission planning task.

15. SUBJECT TERMS

Reinforcement learning, Combinatorial optimization, Scheduling, Vehicle routing, Search, Learning

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Andrew G. Barto
U	U	U	UU	41	19b. TELEPONE NUMBER (Include area code) (413) 545-2109

FINAL TECHNICAL REPORT: F49620-96-1-0254

STOCHASTIC SCHEDULING AND PLANNING USING REINFORCEMENT LEARNING

Principal Investigator: Andrew G. Barto

Abstract–This project investigated the extension of reinforcement learning (RL) methods to large-scale optimization problems relevant to Air Force operations planning, scheduling, and maintenance. The objectives of this project were to: 1) investigate the utility of RL on large-scale logistics problems; 2) extend existing RL theory and practice to enhance the ease of application and the performance of RL on these problems; and 3) explore new problem formulations in order to take maximal advantage of RL methods. A method using RL to modify local search cost functions was developed and shown to yield significant improvement over a traditional local search method on a core vehicle routing problem. A new method for stochastic dynamic optimization was studied, a theoretical result proven, and utility demonstrated using a simulated aerial mission planning task. A learning-based method for optimizing subproblem selection in divide-and-conquer approaches was developed and demonstrated on graph-coloring and on a multiple-vehicle mission planning task.

Executive Summary

Reinforcement learning (RL) is emerging as a promising collection of algorithms for approximating solutions to large-scale stochastic optimization problems that are intractable for conventional exact methods (Bertsekas and Tsitsiklis, 1996; Sutton and Barto, 1998). RL combines stochastic dynamic programming methods for solving Markov decision processes with methods developed by artificial intelligence and neural network researchers. This project investigated the extension of RL methods to large-scale optimization problems relevant to Air Force operations planning, scheduling, and maintenance. The specific objectives of this project were: 1) to investigate the utility of RL on large-scale logistics problems; 2) to extend existing RL theory and practice to enhance the ease of application and the performance of RL on these problems; and 3) to explore new problem formulations in order to take maximal advantage of RL methods.

Many planning and scheduling problems are customarily formulated as discrete optimization problems. For most large-scale discrete optimization problems, one must settle for suboptimal solutions that can be discovered in a reasonable amount of time. The idea was investigated that when many different instances of a problem class have to be solved routinely, an optimization algorithm can actually *learn* from its experiences how to obtain better solutions to new instances of the problem, and how to obtain them more efficiently. The end result of learning is an enhanced optimization algorithm specifically tailored to perform efficiently on the class of problems that supplied the training experiences. To accomplish this, one views local search as a stochastic optimal control problem on which performance can be improved by applying RL. The standard hill-climbing function—the standard cost measure of a problem—is replaced with a cost function that has been learned. Significant decreases in final solution cost are possible using this approach.

Several formulations of this approach were developed and evaluated using the following vehicle-routing problem. N packages (supplies, people, etc.) need to be transported from their current locations to desired destinations by a single vehicle. The vehicle must be routed so that it starts from its base, visits all pick-up and drop-off sites, and returns to the base, with the obvious constraint that a package has to be picked up before it can be dropped off. This tour should be of minimal length. It is assumed that the distances between pick-up and drop-off locations are represented by a symmetric Euclidean distance matrix. This is commonly known as the Dial-A-Ride Problem, or DARP, and is a core vehicle routing problem that often plays a role in more realistic logistics scenarios.

A number of algorithms were compared on DARP for a variety of sizes (number of sites). The best-performing algorithm resulting from learning produced significantly better results than its natural competitors. It is somewhat involved to compare various algorithms of this type since their utility depends on a combination of how long they are allowed to run and the cost of the best tour they find in that time. Given more running time, any algorithm can usually find a lower-cost tour. To provide an overview of the results, however, consider tours that take place within a square with sides of 100 miles with the base in the middle. From results in the literature, for any given number

Markov decision process (MDP) taking actions at a much finer time scale. By taking both perspectives on a single system one can analyze it at higher levels, yet still make changes at the lowest levels. Options permit planning and learning simultaneously at a variety of times scales, and toward a variety of subtasks, substantially increasing the efficiency and abilities of RL systems.

Using this framework, a method was developed that has the potential to be effective in a variety planning and re-planning problems. As an illustration, an abstract mission planning scenario was studied that was motivated by the very difficult problems associated with the need to dynamically re-plan a mission while it is in progress due to the occurrence of incompletely predictable events. A mobile sensor platform, e.g., a uninhabited aerial vehicle (UAV), with limited fuel takes off from its base to observe as many as possible of a set of sites, subject to random fluctuations in the availability of the sites for observation. For example, the sites may be obscured by clouds or other weather phenomena that are only partly predictable and that may change during the mission. As conditions change, the new conditions are made available to the UAV, which must rapidly re-plan the remaining portion of the mission to take them into account. The overall task is to fly from a base, observe as many designated sites as possible, and then return to the base without running out of fuel.

A discrete-event simulation of this task was implemented and a variety of computational experiments were conducted to compare various re-planning methods. The state of the system is described by several variables: the current location of the UAV, its fuel level, the sites it has observed so far, and the current weather at each of the remaining unobserved sites. The primitive control actions are directions of movement (there is no inertia). Even in this very simplified formulation, the state-action space has approximately 24.3 billion elements (assuming 100 discretization levels of the continuous variables) and is intractable by normal stochastic dynamic programming methods. On the other hand, if instead of the low-level primitive directional actions, the action set consists of six high-level actions—options—to fly directly to one of the sites (including the base), the resulting problem (an SMDP) has only 874,800 elements for which it is feasible to compute an optimal policy using standard stochastic dynamic programming. Unfortunately, such a policy is based on executing each selected option to completion so that decisions can only be made when a site is reached. Clearly, the ability to make decisions at a finer time scale can produce better performance, but, as pointed out above, it is intractable to compute an optimal policy in the primitive actions.

The method developed here, called *termination improvement*, is able to perform the feasible planning at the option level (by solving the SMDP), but during execution, continually *re-evaluates* the options at a fine time scale using the value function obtained in solving the SMDP. Suppose at time t the UAV is in the process of executing an option o, e.g., is flying to a particular site s. In re-evaluating the options at time t (based on the state at t), suppose another option, o', evaluates higher than o. Under termination improvement, the UAV would immediately switch to o', i.e., begin flying toward another site, whereas the optimal option-level policy would have it continue flying to s. It is pos-

for algorithm development for both deterministic and stochastic combinatorial optimization.

In summary, these results demonstrate that the idea of enhancing optimization procedures through the use of reinforcement learning and other machine learning techniques has significant promise as a means for improving optimization performance.

Publications Resulting from this Project:

McGovern, A., Moss, E., Barto, A. G. (1999). "Basic-block Instruction Scheduling Using Reinforcement Learning and Rollouts." Proceedings of the International Joint Conference on Artificial Intelligence 1999 (IJCAI99) workshop on Statistical Machine Learning for Large-Scale Optimization, Stockholm, July 1999.

McGovern, A., Moss, E., Barto, A. G. (in press). "Scheduling Straight-Line Code Using Reinforcement Learning and Rollouts." Machine Learning special issue on Reinforcement Learning.

McGovern, A., Precup, D., Ravindran, B., Singh, S., and Sutton, R. S. (1998). "Hierarchical Optimal Control of MDPs." In *Proceedings of the Tenth Yale Workshop on Adaptive and Learning Systems*, pp. 186–191.

Moll, R., Perkins, T., Barto, A. G. (1999). "Enhancing Discrete Optimization with Reinforcement Learning: Case Studies Using DARP." In *Proceedings of the International Joint Conference on Artificial Intelligence 1999 (IJCAI99) workshop on Statistical Machine Learning for Large-Scale Optimization*, Stockholm, July 1999.

Moll, R., Barto, A. G., Perkins, T. J., and Sutton, R. S. (1999). Learning instance-independent value functions to enhance local search. In *Advances in Neural Information Processing Systems: Proceedings of the 1998 Conference.* MIT Press, pp. 1017–1023.

Moll, R., Perkins, T., Barto, A. G. (2000). "Machine Learning for Subproblem Selection." In *Proceedings of the Seventeenth International Conference on Machine Learning*, P. Langley, ed., Morgan Kaufmann, San Francisco CA, pp. 615622.

Precup, D., Sutton, R. S. (1998). "Multi-time Models for Temporally Abstract Planning." In *Advances in Neural Information Processing Systems: Proceedings of the 1997 Conference.* MIT Press, pp. 1050–1056.

Precup, D., Sutton, R. S., Singh, S. P. (1998). "Theoretical Results on Reinforcement Learning with Temporally Abstract Options." In *Proceedings of the Tenth European Conference on Machine Learning.* Springer-Verlag. pp. 382–393.

Randlov, J., Barto, A. G., Rosenstein, M. T. (2000). "Combining Reinforcement Learning with a Local Control Algorithm." In *Proceedings of the Seventeenth International Conference on Machine Learning*, P. Langley, ed., Morgan Kaufmann, San Francisco CA, pp. 775–782.

1 Learning Instance-Independent Value Functions to Enhance Local Search

Combinatorial optimization is of great importance in computer science, engineering, and operations research. We investigated the use of RL to enhance traditional local search optimization (hillclimbing). Since local search is a sequential decision process, RL can be used to improve search performance by learning an evaluation function that predicts the outcome of search and is therefore able to guide search to low-cost solutions better than can the original cost function.

Three approaches to using RL to improve combinatorial optimization have been described in the literature. One is to learn a value function over multiple search trajectories of a single problem instance. As the value function improves in its predictive accuracy, its guidance enhances additional search trajectories on the same instance. Boyan and Moore's STAGE algorithm (Boyan and Moore 1997, Boyan 1998) falls into this category, showing excellent performance on a range of optimization problems. Another approach is to learn a value function off-line and then use it over multiple new instances of the same problem. Zhang and Dietterich's (1995) application of RL to a NASA space shuttle mission scheduling problem takes this approach (although it does not strictly involve local search as we define it below). A key issue here is the need to normalize state representations and rewards so that trajectories from instances of different sizes and difficulties yield consistent training data. In each of the above approaches, a state of the RL problem is an entire solution (e.g., a complete tour in a Traveling Salesman Problem (TSP)) and the actions select next solutions from the current solutions' neighborhoods. A third approach, described by Bertsekas and Tsitsiklis (1996), uses a learned value function for guiding the direct construction of solutions rather than for moving between them.

We first focused on combining aspects of first two of these approaches with the goal of carefully examining how well the TD(λ) algorithm (see Sutton and Barto, 1998) can learn an instance-independent value function for a given problem to produce an enhanced local search algorithm applicable to all instances of that problem. Our approach combines an off-line learning phase with STAGE's alternation between using the learned value function and the original cost function to guide search. We studied this algorithm's application to a somewhat complicated variant of TSP known as the Dial-A-Ride Problem (DARP), which exhibits some of the non-uniform structure present in real-world transportation and logistics problems. We then examined the use of "rollouts" (Tesauro and Galperin, 1996; Bertsekas *et al.*, 1997) to enhance a constructive DARP algorithm due to Kubo and Kasugai (1990) (Section 1.5). We also proposed and experimented with a new algorithm which we call the *expected improvement* algorithm (Section 1.6). It uses the same control structure as Boyan's (1998) STAGE but learns a different hill-climbing function—one that seeks to maximize the expected improvement over the best-so-far solution, rather than just the expected value of hill-climbing. We report results obtained with this algorithm on DARP.

Dial-A-Ride problem.

1.2 The Dial-a-Ride Problem

The Dial-a-Ride Problem (DARP) has the following formulation. A van is parked at a terminal. The driver receives calls from N customers who need rides. Each call identifies the location of a customer, as well as that customer's destination. After the calls have been received, the van must be routed so that it starts from the terminal, visits each pick-up and drop-off site in some order, and then returns to the terminal. The tour must pick up a passenger before eventually dropping that passenger off. The tour should be of minimal length. Failing this goal—and DARP is NP-complete, so it is unlikely that optimal DARP tours will be found easily—at least a good quality tour should be constructed. We assume that the van has unlimited capacity and that the distances between pick-up and drop-off locations are represented by a symmetric Euclidean distance matrix.

We use the notation

$$0\ 1\ 2\ -1\ 3\ -3\ -2$$

to denote the following tour: "start at the terminal (0), then pick up 1, then 2, then drop off 1 (thus: -1), pick up 3, drop off 3, drop off 2 and then return to the terminal (site 0)." Given a tour s, the *2-opt neighborhood* of s, $A_2(s)$, is the set of legal tours obtainable from s by subsequence reversal. For example, for the tour above, the new tour created by the following subsequence reversal

$$0\ 1\ /\ 2\ -1\ 3\ /\ -3\ -2 \longrightarrow 0\ 1\ 3\ -1\ 2\ -3\ -2$$

is an element of $A_2(T)$. However, this reversal

$$0\ 1\ 2\ /\ -1\ 3\ -3\ /\ -2 \longrightarrow 0\ 1\ 2\ -3\ 3\ -1\ -2$$

leads to an infeasible tour, since it asserts that passenger 3 is dropped off first, then picked up. The neighborhood structure of DARP is highly non-uniform, varying between A_2 neighborhood sizes of $O(N)$ and $O(N^2)$.

Let s be a feasible DARP tour. By 2-opt(s) we mean the tour obtained by first-improvement local search using the A_2 neighborhood structure (presented in a fixed, standard enumeration), with tour length as the cost function. As with TSP, there is a 3-opt algorithm for DARP, where a 3-opt neighborhood $A_3(s)$ is defined and searched in a fixed, systematic way, again in first-improvement style. This neighborhood is created by inserting three rather than two "breaks" in a tour. 3-opt is much slower than 2-opt, more than 100 times as slow for $N = 50$, but it is much more effective, even when 2-opt is given equal time to generate multiple random starting tours and then complete its improvement scheme.

Psaraftis (1983) was the first to study 2-opt and 3-opt algorithms for DARP. He studied tours up to size $N = 30$, reporting that at that size, 3-opt tours are about 30%

Table 1: Weight Vectors for Learned Value Functions.

Value Function	Weight Vector
V	$< .951, .033, .0153 >$
V_{20}	$< .981, .019, .00017 >$
V_{30}	$< .984, .014, .0006 >$
V_{40}	$< .977, .022, .0009 >$
V_{50}	$< .980, .019, .0015 >$
V_{60}	$< .971, .022, .0069 >$

bias weight and features developed from the following base features: 1) $normcost_N(s) = c(s)/Stein_N$; 2) $normhood_N = |A(s)|/a_N$, where a_N is a normalization coefficient defined below; and 3) $normprox_N$, which considers a list of the $N/4$ least expensive edges of the distance matrix, as follows. Let e be one of the edges, with endpoints u and v. The $normprox_N$ feature examines the current tour, and counts the number of sites on the tour that appear between u and v. $normprox_N$ is the sum of these counts over the edges on the proximity list divided by a normalizing coefficient b_N described below. Our function approximator is then given by $w_0 + normcost_N/(normhood_N)^2 w_1 + normprox_N/(normhood_N)^2 w_2$. The coefficients a_N and b_N are the result of running linear regression on randomly sampled instances of random sizes to determine coefficients that will yield the closest fit to a constant target value for normalized neighborhood size and proximity. The results were $a_N = .383N^2 + .28.5N - 244.5$ and $b_N = .43N^2 + .736N - 68.9\sqrt{N} + 181.75$. The motivation for the quotient features comes from Healy and Moll (1995) who found that using a similar term improved 2-opt on DARP by allowing it to sacrifice cost improvements to gain large neighborhoods.

1.4 Experimental Results

Comparisons among algorithms were done at five representative sizes $N = 20$, 30, 40, 50, and 60. For the learning phase, we conducted approximately 3,000 learning episodes, each one using a randomly generated instance of size selected randomly between 20 and 60 inclusive. The result of the learning phase was a value function V. To assess the influence of this multi-instance learning, we also repeated the above learning phase 5 times, except that in each we held the instance size fixed to a different one of the 5 representative sizes, yielding in each case a distinct value function V_N, where N is the training instance size. Table 1 shows the resulting weight vector < bias weight, $costhood_N$ weight, $proximity_N$ weight >. With the exception of the $proximity_N$ weight, these are quite consistent across training instance size. We do not understand why training on multiple-sized instances led to this pattern of variation.

Table 4: Performance Comparisons, Equalized for Running Time.

	Size and Running Time				
	N=20	N=30	N=40	N=50	N=60
Algorithm	10 sec	20 sec	40 sec	100 sec	150 sec
2-opt	16	29	28	30	38
STAGE	18	20	32	24	27
TD(.8) $\epsilon = 0$	12	13	16	22	20
TD(.8) $\epsilon = .01/N$	13	11	14	24	28

time, including the STAGE algorithm using linear regression with our features. We generated 20 random instances at each of the representative sizes, and we allowed each algorithm to run for the indicated amount of time on each instance. If time remained when a local optimum was reached, we restarted the algorithm at that point, except in the case of 2-opt, where we selected a new random starting tour. The restarting regime for the learning-enhanced algorithms is the regime employed by STAGE. Each algorithm reports the best result found in the allotted time, and the chart reports the averages of these values across the 20 instances. Notice that the algorithms that take advantage of extensive off-line learning significantly outperform the other algorithms, including STAGE, which relies on single-instance learning.

1.5 Enhancing a Constructive Method using Rollouts

Here we report results using rollouts in DARP, following the approach taken by Tesauro and Galperin (1996) to backgammon and by Bertsekas *et al.* (1997) to combinatorial optimization. A rollout is a simulation of the consequences of a decision that provides an estimate of its effectiveness. In a stochastic setting, averaging the results of many rollouts provides a Monte Carlo estimate of effectiveness that can be used to direct decision making. We used rollouts to enhance a constructive algorithm for DARP in which tours are constructed by sequentially inserting pairs of sites (a pick-up site and its corresponding drop-off site). At each iteration of a constructive algorithm, a site pair is selected from those not already in the partial tour. This pair is then inserted into the partial tour according to some rule. If we assume that the rule for inserting a pair is given, then the algorithm has to decide which pair to select at each iteration. We used rollouts to make these decisions. For each candidate pair, the current partial tour is completed by first inserting that pair according to the given rule and then completing the schedule using some heuristic decision process. This constitutes a rollout. Only one rollout is needed for each candidate pair since the construction process is deterministic. Then the pair whose rollout led to the shortest tour is inserted, and the process repeats from the resulting partial tour until a complete tour is obtained. Bertsekas *et al.* (1997) showed that the resulting algorithm is always better than or equal to the heuristic decision process it uses

Table 6: Performance of the Rollout Method based on PI-Farthest for Sizes $N = 20$ and 30. Entries are percentage above $Stein_N$ averaged over 100 random instances of size N.

	N=20	N=30
min	-11.18	-10.77
mean	1.84	1.63
max	15.33	15.56
std	5.04	3.91
time (per tour)	98 sec	756 sec

benefit was gained when we used rollouts to select the *first* site pair to insert, with the benefit decreasing as rollouts were used to make later decisions. Note, however, that using rollouts for later decisions takes less time than for earlier decisions since each rollout is shorter. Doing rollouts at the start to make the first decision has time complexity $O(n^4)$ while still showing a significant improvement over pi-farthest at size $n = 20$, but we did not test this for larger n.

Table 7: Performance with Rollouts used for the First Decision Only for Size $N = 20$. Entries are percentage above $Stein_N$ averaged over 100 random instances of size N.

	N=20
min	-15.05
mean	3.26
max	16.11
std	6.20
time (per tour)	10 sec

We also performed a number of learning experiments where the goal was to learn state-action values (Q-values) for pair selection. We tried numerous variations, including different feature sets and function approximation methods. Some of these methods learned to perform as well as PI-farthest (in fact some learned to do the same thing as PI-farthest), but no results were particularly better than PI-farthest. There is much more that could be done here.

1.6 The Expected Improvement Algorithm

Here we describe the *expected improvement algorithm*, an algoritm we designed in an attempt to improve over Boyan's (1998) STAGE algorithm. Although our experimental

1. Start a new trajectory from a random solution in the set of feasible solutions. Call this starting point S_0.

2. Hill-climb on the cost function to a local optimum. Suppose hill-climbing traverses solutions $S_1, \ldots, S_n$ and arrives at a solution with cost C.

3. Add the cost hill-climbing data to the training data and retrain the learned hill-climbing function. For STAGE this means adding the training pairs $S_i \rightarrow C$ to the data set and refitting the data, e.g. by polynomial regression. Learning the expected improvement function from a set of hill-climbing data is described below.

4. Hill-climb on the learned function to a local optimum.

5. If learned function hill-climbing went anywhere, i.e., did not start at a local optimum, then continue with cost hill-climbing (step 2). Otherwise, start a new trajectory (step 1).

The learning data is accumulated from all cost hill-climbing, across trajectories. An additional complication not listed in the algorithm is a *patience* parameter. In local search, the number of solutions adjacent to a particular solution can often be very large. Near optima of the cost function, typically only a very few neighbors will improve on the cost of a current solution and the improvement will be small anyway. It is thus sometime more practical to examine a random subset of the neighbors of a solution. If no better neighbors are found in that subset, the algorithm acts as if it were truly a local optimum and proceeds to the next step. The patience parameter, a fraction in $(0, 1]$, gives the proportion of the neighborhood that must be examined before a solution is deemed a local optimum. Patience= 1 corresponds to searching the entire neighborhood.

We could try to learn $EI(s, b)$ in typical supervised-learning fashion. At any point during cost hill-climbing we are at some solution s and have best-so-far solution b. Cost hill-climbing from s yields some outcome x, which we use to construct a single training sample for the EI function:

$$(s, b) \rightarrow \begin{cases} b - x & \text{if } b > x \\ 0 & \text{if } b \leq x. \end{cases}$$

However, this makes rather inefficient use of the hill-climbing information. Once we have a reasonably good solution, the observed improvements will nearly always be zero. Further, the observed hill-climbing outcome x could equally well be used to update our estimate of EI for any $b' \neq b$, because the outcome of hill-climbing is independent of b. We are currently investigating a technique that would allow us to update EI for all b', but take a simpler approach in this paper.

We assume that the outcome of cost hill-climbing from a state, $CH(s)$ has a Gaussian distribution. We use the cost hill-climbing data to learn estimates for the first and second moments of $CH(s)$, which determines the mean (μ) and standard deviation (σ) of the

Table 8: Expected Improvement Algorithm Results on DARP: Best Solution Statistics Listed as Percentage Over Stein Estimate

patience = 1.0 instances = 1...25

	mean	max	min	std	2*std/$\sqrt{25}$
S/LR	6.88	48.02	-7.14	14.51	5.80
S/QR	16.74	65.71	-0.66	16.89	6.75
EI/LR	20.60	32.90	8.40	6.29	2.51
EI/QR	7.82	29.83	-3.22	8.01	3.20

patience = 0.9 instances = 1...25

	mean	max	min	std	2*std/$\sqrt{25}$
S/LR	4.78	21.42	-4.08	6.57	2.62
S/QR	15.03	26.04	0.0	7.03	2.81
EI/LR	15.49	30.50	0.10	8.63	3.45
EI/QR	8.03	13.74	-1.52	4.46	1.78

patience = 0.5 instances = 1...22

	mean	max	min	std	2*std/$\sqrt{22}$
S/LR	0.65	9.57	-4.70	3.66	1.56
S/QR	8.00	20.32	0.44	4.87	2.07
EI/LR	22.28	34.49	11.76	6.01	2.56
EI/QR	8.77	16.21	-1.18	5.13	2.18

patience = 0.2 instances = 1...30

	mean	max	min	std	2*std/$\sqrt{30}$
S/LR	3.94	15.05	-4.66	4.59	1.67
S/QR	10.32	19.78	0.077	4.97	1.81
EI/LR	25.44	39.37	15.31	6.48	2.36
EI/QR	12.21	26.97	1.39	5.84	2.13

2 Planning and Re-Planning with Temporal Abstraction

Fundamental to the theory of systems and control is the problem of representing knowledge about the environment and about possible courses of action at a multiplicity of interrelated temporal scales. For example, a human traveler must decide which cities to go to, whether to fly, drive, or walk, as well as the individual muscle contractions involved in each step. In military planning and execution, decisions are obviously made at different levels of temporal detail at different levels in the chain of command. Appropriate temporal abstractions can enable complex problems to be represented at a higher level that involves fewer states, fewer choices, and fewer steps. Formulating such abstractions has been the objective of a large and diverse body of work in control systems, artificial intelligence, and operations research.

We made significant progress in formulating and solving problems of temporal abstraction within the framework of RL. We formulated a generic concept of temporally extended "courses of action," called *options*, which includes both primitive control actions as well as temporally extended courses of actions. Options are essentially the same as the variable-duration actions of a semi-Markov decision process (SMDP), but they are superimposed on a a low-level Markov decision process (MDP) taking whose actions are at a much finer time scale. By taking both perspectives on a single system we are able to analyze it at higher levels, yet still make changes at the lowest levels. Our formulation enables options to be used in place of actions in all planning and learning processes conventionally used in RL and dynamic programming. Options and models of options can be learned for a wide variety of different subtasks, and then rapidly combined to solve new tasks. Options permit planning and learning simultaneously at a variety of times scales, and toward a variety of subtasks, substantially increasing the efficiency and abilities of RL systems.

We illustrated this result in an abstract reconnaissance mission re-planning scenario. An aerial sensor platform with limited fuel takes off from its base to observe as many as possible of a set of sites, subject to random fluctuations in the availability of the sites for observation. For example, the sites may be obscured by clouds or other weather phenomena that are only partly predictable and that may change during the mission. As conditions change, the remaining portion of the mission must be rapidly re-planned to take into account the new conditions. Because of the large state space and the stochasticity, even moderate-sized instances of this problem are intractable by conventional methods such as dynamic programming, or even conventional RL. We have shown that by introducing options in the form of controllers for flying directly to each site, the problem can be made vastly easier and solved at the SMDP level with modest computational effort. When this solution is then used at the MDP level for real-time control, performance far better than receding-horizon methods is obtained.

The input set and termination condition of an option together limit the states over which the option's policy needs to be defined. For example, a handcrafted policy π for a mobile robot to dock with its battery charger might be defined only for states $\mathcal{I}$ in which the battery charger is within sight. The termination condition β could be defined to be 1 outside of $\mathcal{I}$ and when the robot is successfully docked.

We can now define *policies over options*. Let the set of options available in state s be denoted $\mathcal{O}_s$; the set of all options is denoted $\mathcal{O} = \bigcup_{s \in \mathcal{S}} \mathcal{O}_s$. When initiated in a state s_t, the Markov policy over options $\mu : \mathcal{S} \times \mathcal{O} \mapsto [0,1]$ selects an option $o \in \mathcal{O}_{s_t}$ according to the probability distribution $\mu(s_t, \cdot)$. The option o is then taken in s_t, determining actions until it terminates in s_{t+k}, at which point a new option is selected, according to $\mu(s_{t+k}, \cdot)$, and so on. In this way a policy over options, μ, determines a (non-stationary) policy over actions, or *flat policy*, $\pi = f(\mu)$. We define the value of a state s under a general flat policy π as the expected return if the policy is started in s:

$$V^\pi(s) \stackrel{\text{def}}{=} E\Big\{r_{t+1} + \gamma r_{t+2} + \cdots \Big| \mathcal{E}(\pi, s, t)\Big\},$$

where $\mathcal{E}(\pi, s, t)$ denotes the event of π being initiated in s at time t. The value of a state under a general policy (i.e., a policy over options) μ can then be defined as the value of the state under the corresponding flat policy: $V^\mu(s) \stackrel{\text{def}}{=} V^{f(\mu)}(s)$. An analogous definition can be used for the *option-value* function, $Q^\mu(s, o)$.

2.2 SMDP Planning

Options are closely related to the actions in a decision problem known as a *semi-Markov decision process* (SMDP). In fact, any MDP with a fixed set of options *is* an SMDP. Accordingly, the theory of SMDPs provides an important basis for the theory of options. We briefly review the standard SMDP framework for planning, which will provide the basis for our extension.

Planning with options requires a model of their consequences. The standard form of this model is given in the theory of SMDPs. The reward part of the model of o for state $s \in \mathcal{S}$ is the total reward received along the way:

$$r_s^o = E\Big\{r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{k-1} r_{t+k} \Big| \mathcal{E}(o, s, t)\Big\},$$

where $\mathcal{E}(o, s, t)$ denotes the event of o being initiated in state s at time t. The state-prediction part of the model is

$$p_{ss'}^o = \sum_{j=1}^{\infty} \gamma^j \Pr\{s_{t+k} = s', k = j \mid \mathcal{E}(o, s, t)\} = E\Big\{\gamma^k \delta_{s' s_{t+k}} \mid \mathcal{E}(o, s, t)\Big\},$$

for all $s' \in \mathcal{S}$, under the same conditions, where $\delta_{ss'}$ is an identity indicator, equal to 1 if $s = s'$, and equal to 0 else. We call this kind of model a *multi-time model* because it

compare the value of continuing with o, which is $Q^\mu(s_t, o)$, to the value of terminating o and selecting a new option according to μ, which is $V^\mu(s) = \sum_{o'} \mu(s, o') Q^\mu(s, o')$. If the latter is more highly valued, then why not terminate o and allow the switch? We proved that this new way of behaving is indeed better (Sutton *et al.*, 1999), but this is a change in the termination condition of o and thus requires stepping outside the existing set of options.

We can characterize the new way of behaving as following a policy μ' that is the same as the original one, but over new options, i.e., $\mu'(s, o') = \mu(s, o)$, for all $s \in \mathcal{S}$. Each new option o' is the same as the corresponding old option o except that it terminates whenever termination seems better than continuing according to Q^μ. We call such a μ' a *termination improved policy* of μ. We will now state a general theorem, which extends the case described above, in that options have to be semi-Markov (instead of Markov) and termination improvement is optional at each state where it could be done. This lifts the requirement that Q^μ be completely known.

Theorem 1 (Termination Improvement) *For any MDP, any set of options $\mathcal{O}$, and any Markov policy $\mu : \mathcal{S} \times \mathcal{O} \mapsto [0, 1]$, define a new set of options, $\mathcal{O}'$, with a one-to-one mapping between the two option sets as follows: for every $o = \langle \mathcal{I}, \pi, \beta \rangle \in \mathcal{O}$ we define a corresponding $o' = \langle \mathcal{I}, \pi, \beta' \rangle \in \mathcal{O}'$, where $\beta' = \beta$ except that for any history h in which $Q^\mu(h, o) < V^\mu(s)$, where s is the final state of h, we may choose to set $\beta'(h) = 1$. Any histories whose termination conditions are changed in this way are called termination-improved histories. Let μ' be the policy over o' corresponding to μ: $\mu'(s, o') = \mu(s, o)$, where o is the option in $\mathcal{O}$ corresponding to o', for all $s \in \mathcal{S}$. Then*

1. *$V^{\mu'}(s) \geq V^\mu(s)$ for all $s \in \mathcal{S}$.*
2. *If from state $s \in \mathcal{S}$ there is a non-zero probability of encountering a termination-improved history upon initiating μ' in s, then $V^{\mu'}(s) > V^\mu(s)$.*

As one application of this result, consider the case in which μ is an optimal policy for a given set of Markov options $\mathcal{O}$. By planning or learning we can determine the SMDP optimal value function $V^*_{\mathcal{O}}$ and the optimal policy $\mu^*_{\mathcal{O}}$ that achieves it. This is indeed the best that can be done without changing $\mathcal{O}$, that is, in the SMDP defined by $\mathcal{O}$, but less than the best possible achievable in the MDP, which is V^*. But of course we typically do not wish to work directly in the primitive options because of the computational expense. The termination improvement theorem gives us a way of improving over $\mu^*_{\mathcal{O}}$ with very little additional computational expense, by stepping outside $\mathcal{O}$. The only additional expense is the cost of checking (on each time step) if a better option exists, which is negligible compared to the combinatorial process of computing $Q^*_{\mathcal{O}}$.

while picking only from $\mathcal{O}$. The optimal policy within $\Pi(\mathcal{O})$ runs from landmark to landmark, as shown by the thin line in Figure 1. This is the optimal solution to the SMDP defined by $\mathcal{O}$ and is indeed the best that one can do while picking only from these options. But of course one can do better if the options are not followed all the way to each landmark. The trajectory shown by the thick line in Figure 1 cuts the corners and is shorter. This is the termination-improved policy with respect to the SMDP-optimal policy. The termination-improved policy takes 474 steps from start to goal which, while not as good as the optimal policy in primitive actions (425 steps), is much better than the SMDP-optimal policy, which takes 600 steps. The state-value functions, $V^{\mu_{\mathcal{O}}^*}$ and $V^{\mu'}$ for the two policies are also shown in Figure 2.

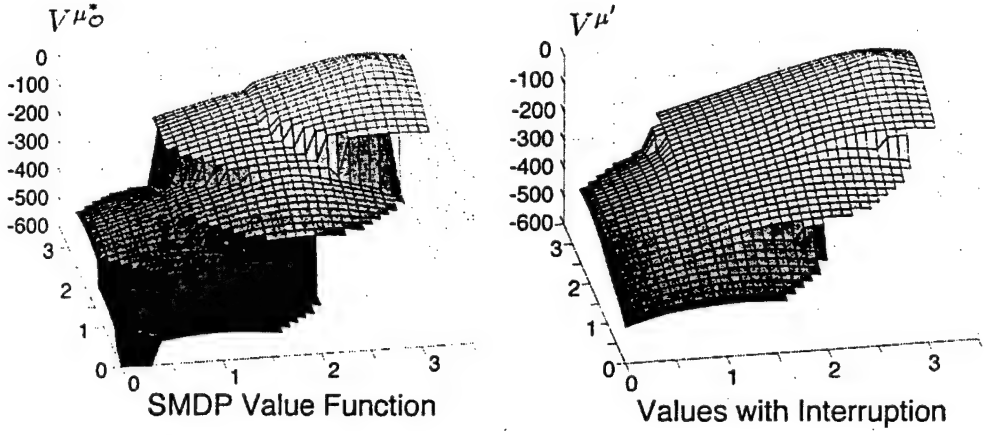

Figure 2: Termination improvement in navigating with landmark-directed controllers. The state-value functions for the SMDP-optimal and termination-improved policies. Note that the latter is greater.

2.4.2 Aerial Surveillance Mission Planning Task

Figure 3 presents a more complex mission planning task which we call the Uninhabited Aerial Vehicle (UAV) task since it was intended to represent some aspects of mission planning for this technology. A mission is a flight from base to observe as many sites as possible, from a given set of sites, and return to base without running out of fuel. The local weather at each site toggles from cloudy to clear according to independent Poisson processes. If the sky at a given site is cloudy when the plane gets there, no observation is made and the reward is 0. If the sky is clear, the UAV gets a reward, according to the importance of the site. The positions, rewards, and mean time between two weather changes for each site are given in Figure 3. The UAV has a limited amount of fuel, and it consumes one unit of fuel during each time tick. If the fuel runs out before reaching the base, the UAV crashes and receives a reward of -100.

The primitive actions are small movements in any direction (there is no inertia). The state of the system is described by several variables: the current position of the UAV,

3 Learning Subproblem Selection Techniques for Combinatorial Optimization

Divide and conquer—that is, subproblem generation, solution, and recombination—is a standard technique in combinatorial optimization. In many divide and conquer settings the search for suitable subproblems is itself a significant component of the technique, and overall results can depend critically on the component subproblems that are actually considered. For example, in a classical vehicle routing problem in Operations Research, a fleet of vans, each with a fixed carrying capacity and each starting from a common depot, is given the job of making deliveries to a set of sites and then returning to the depot. The deliveries assigned to any van must not exceed that van's carrying capacity. The problem objective is to assign deliveries to the vans so as to respect the capacity constraints, and to route vans so as to minimize the total van travel distance. Clearly, how sites are assigned to vans, i.e., which single van subproblems are formed, is at least as important as the routing plan for each van.

How a problem is factored into subproblems, e.g., which sites are assigned to which vans, is often done heuristically. Here we describe an alternative approach to subproblem selection, one that uses machine learning to direct search in the space of problem decompositions. While in principle we do not restrict the form of this search, in both of the examples presented here we use local search over suitably formulated spaces of subproblems.

Suppose an optimization problem P, such as the vehicle routing problem described above, can be solved by first decomposing it into subproblems from some class C, and then by solving those subproblems. In the example above the class C consists of single van routing problems. Now suppose algorithm A solves instances from C. Instead of factoring P into subproblems according to some heuristic rule, as is common practice, we view the task of choosing a decomposition as a search problem in the space of subproblem decompositions. The effectiveness of this search depends on being able to estimate algorithm A's performance quickly on example decompositions. To accomplish this we first identify a set of quickly computable features that capture, with some precision, the expected behavior of A on any instance. Then, using supervised learning in an off-line training procedure, we create $\hat{A}(\cdot)$, an estimate of A's performance. This is done by running A on a large corpus of examples, and then using regression to fit the feature values of the examples to A's corresponding output values. Once constructed, we use $\hat{A}$—a fast, inexpensive substitute for A itself—as a cost function for hillclimbing in the space of subproblem decompositions. When we arrive at a local optimum in "subproblem decomposition" space, we apply A to the subproblem or subproblems associated with the optimum, and use the results obtained to form an overall problem solution.

We illustrate our technique with two examples, graph coloring for a class of geometric graphs, a deterministic problem, and the Multiple Uninhabited Air Vehicle (MUAV) surveillance problem, a stochastic, traveling salesman-like problem.

2. Using standard first-improvement local search flow of control, hill-climb in the space of recolor sets by repeatedly considering pairwise exchanges between a member of the current recolor set and a member of some still-intact color class S. Exchanges are accepted if the swap preserves the integrity of S (here: the alteration of S should introduce no adjacencies to S, i.e., no vertices in S should be linked by an edge), and if $\hat{A}$'s estimate of A's performance on the recolor set is improved by the exchange.

3. When a local optimum is reached in 2), solve the subproblem by applying A to the newly constituted recolor set.

4. Once this subproblem has been solved, combine its coloring scheme with the other color classes (which may be slightly altered as a result of the exchange process, but which are still legal classes) to get a complete, optimized solution to the problem.

Notice that the above formulation—selecting a subproblem by hill-climbing in subproblem space using a learned estimator for a known algorithm— does not depend on the kind of graph under consideration, and indeed does not depend on the objects being graphs at all. In fact our methodology applies to any constrained partitioning problem and any algorithm A that does a poor job on some of the partitions. For example, the transformation of this methodology to classical one-dimensional bin-packing is quite direct. One-dimensional bin-packing is the problem of packing items of size between 0 and 1 in as few unit-sized bins as possible, such that the items assigned to any bin sum to a value ≤ 1. To lift our graph coloring methodology, start with an approximation algorithm A such as First-Fit-Decreasing (Papadimitriou and Steiglitz, 1982), and build an estimator, $\hat{A}$, for it. Apply A to pack the bins. Collect some subset of poorly packed bins and empty their contents, forming a repacking set. Now repeatedly exchange elements of this set for items that still reside in bins. An exchange is accepted if it maintains the integrity of the still intact bin (the sum of that bin's contents remains ≤ 1 after the exchange), and if $\hat{A}$'s estimate of the repacking set improves as a result of the exchange. When this process reaches a local optimum, apply A to this final repacking set, and combine these bins with the previously packed bins to obtain a complete solution. In section 4 we briefly consider a multi-knapsack packing problem, which illustrates some complications that arise with our model.

3.1.1 Graph Coloring Results

We considered geometric graphs from the classes $U_{N,t}$ for $N = 250$, 500, and 750 vertices, and for threshold $t = .5$. We use the DSATUR algorithm as our base graph coloring algorithm (Brelaz, 1979). This algorithm is reported to be the best, or at least competitive with the best algorithms known for geometric graphs (Johnson *et al.*, 1991). It works as follows:

Table 9: Frequency of Optimization Improvement over DSATUR

Algorithm / Problem Size	N = 250	N = 500	N = 750
no swapping	14%	22%	23%
full optimization, RecolorClassCt = 10	56%	69%	75%
full optimization, RecolorClassCt = 15	57%	77%	80%

Table 10: Optimization Routine Running Times, DSATUR = 1.0

Algorithm / Problem Size	N = 250	N = 500	N = 750
full optimization, RecolorClassCt = 10	1.3	1.7	1.8
full optimization, RecolorClassCt = 15	2.6	3.6	7.1

Table 10 shows the running time of the algorithm at various graph sizes and different settings of RecolorClassCt. Results are given in terms of a base running time of 1.0, at each size, for DSATUR.

We also compared the performance of our enhanced DSATUR algorithm against pure DSATUR on a time-equalized basis. For this comparison we set $RecolorClassCt = 10$ since our optimization scheme is much faster at this setting, while performance is only slightly diminished. We compared these algorithms on 100 randomly generated examples at three sizes. For each example we ran DSATUR from 10 random starting orders. We recorded the best coloring found among the 10, and we also recorded the running time for the 10 runs. We then allowed our enhanced DSATUR algorithm to run for an equivalent amount of time. Table 11 gives the percentage of example instances on which our optimized DSATUR algorithm colored graphs using fewer colors.

While our results for $U_{N,.5}$ are encouraging, especially in light of DSATUR's apparent success on this class compared with other algorithms, we should point out that to be effective, the algorithm requires a certain amount of time-consuming elaboration, which first involves choosing appropriate features, and then requires solving several regression problems, namely feature normalization and learning $\hat{A}$ off-line. Thus even a class as closely related to $U_{N,.5}$ as $U_{N,.9}$ would require at least several additional hours in order

Table 11: Enhanced DSATUR improvement frequency over pure DSATUR, equalized for time

	N = 250	N = 500	N = 750
full optimization, RecolorClassCt = 10	22%	47%	50%

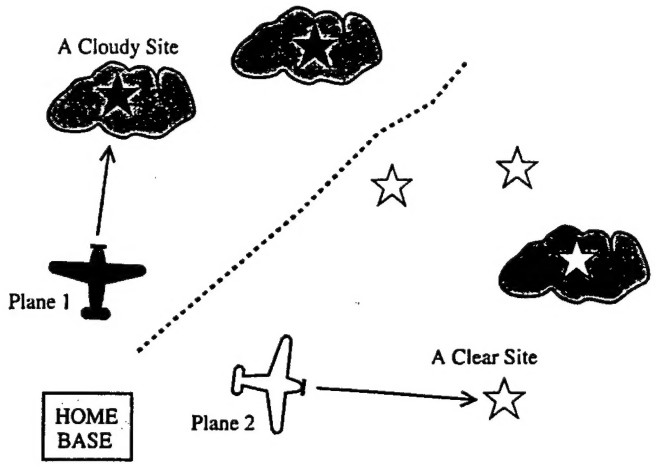

Figure 4: Depiction of an MUAV instance.

alone return) to 5.0 (enough to circumnavigative the unit-area surveillance region with fuel to spare).

We approach this task by splitting the sites among the UAVs, creating a set of single-UAV surveillance tasks. These single-UAV subtasks are solved by a simple heuristic controller, which we found to be the most effective of several obvious contenders. If possible, a UAV flies towards the nearest clear-weather site in its partition. If all the sites are cloudy, the UAV simply flies to the nearest of those. And finally, when a UAV has no assigned sites, it heads back to base. We call this control rule "Nearest Site, Visible Preferred", or just NVPfor short.

The question, then, is how to divide the sites in a MUAV task among the UAVs for solution by NVP. To do this, we first learn $\widehat{\text{NVP}}$, an estimate of the performance of NVPon arbitrary, hypothesized single UAV instances. A local search procedure in the space of partitions, using performance estimator $\widehat{\text{NVP}}$ to evaluate components of a candidate partition, then identifies a promising division of the MUAV instance into a collection of single-plane problems.

To learn $\widehat{\text{NVP}}$, we generated 500 random, single-UAV surveillance problem instances, with 6-10 sites. The number of sites, their placement in the unit square, and initial weather conditions were all chosen uniformly randomly. The UAV was given 3.0 fuel, and was allowed to reevaluate and possibly choose a new heading every 0.025 time units. The home base for the UAV was at the origin, (0,0). The changes in weather at each site occurred at discrete times according to Poisson process with parameter $\lambda = 2.0$.

On each instance, we simulated one run of NVP, recording at each decision point 18 features and the total observation reward achieved from that point forward in the run. We did a linear least-squares fit to predict the fraction of remaining reward that will be achieved. Multiplying by the amount of remaining reward thus predicts the total observation reward. It is this linear approximation that we use to estimate the performance of NVPon potential single-UAV subproblems resulting from the decomposition of an MUAV

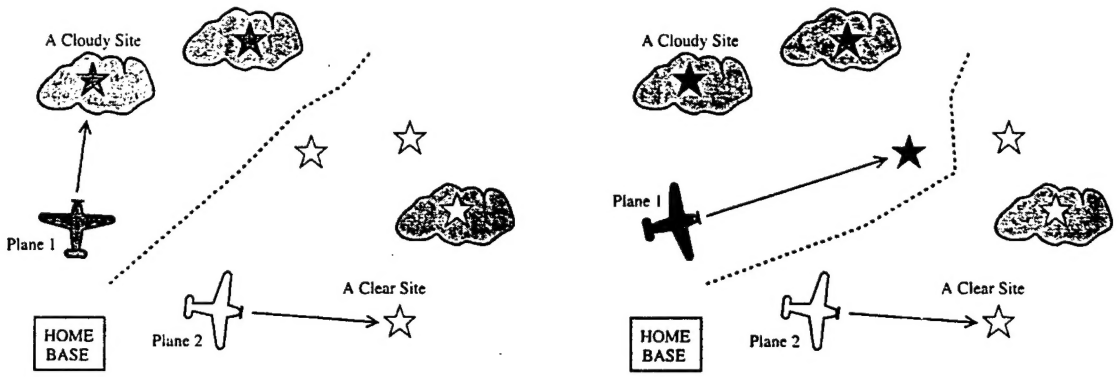

Figure 5: MUAV task: Before (left) and after (right) a single local search step.

partitioning was also reoptimized. We used the same local search procedure, starting from the current partitioning. This allowed the controller of the fleet of UAVs to react to variable performance caused by weather. For example, if one UAV had made little progress because of cloudy weather at its sites while another had made much progress in its partition, then sites might be reassigned to balance current loads among UAVs. The continual optimization even allows the system to respond to changes not modelled as part of the dynamics of the system. For instance, if we were to remove or add UAVs, or change the sites, the system would seamlessly redivide the problem for the individual UAVs.

We compared four different partitioning schemes on 100 random 24-site, 3-plane MUAV instances, with different amounts of fuel. We tried partitioning based on the linear NVPperformance estimates and also partitioning based on compactness—feature 1 in Table 12. Partitioning based exclusively on compactness means that we accepted a single move site reassignment when by doing so we improved (reduced) the sum of the compactness measures for the three groups of sites. Compactness-based partition was the best heuristic method we tested, outperforming sector-based partitioning, a standard approach in multiple-vehicle routing problems. For both, we ran trials in which the initial partitioning was held fixed throughout the run, and trials in which the partitioning was continually optimized.

Table 13 summarizes the results. Within each row, the boldface numbers are statistically significantly larger than the other numbers, by a t-test at $p = 0.05$. Through much of the fuel range studied, optimizing $\widehat{\text{NVP}}$led to better partitioning than did compactness-based partitioning. At the highest levels of fuel, compactness performs a little better. In the single-UAV trials that provided the data for the NVPestimate, the UAV always started with 3.0 fuel. It is possible that adjusting this would change the range where partitioning based on $\widehat{\text{NVP}}$performs best.

Note also that the continual optimization of partitions yielded significant improvements over sticking with initial partitionings. In this domain, redividing the full problem into different subproblems is easy to do: there is little cost in moving a site to another

stochastic problem. That is, the factorization style we employed meant that we needed to construct an estimator function via simulations for the single plane case only. The overall stochastic characteristics of the full MUAV are handled by the resulting single plane estimator, combined with a nonstochastic parititioning algorithm, applied repeatedly to respond to uncertainites as they arise. We believe this methodology holds promise for other, similarly factorable stochastic optimization problems.

References

Boyan, J. A. (1998). Learning Evaluation Functions for Global Optimization. Ph.D. Thesis, Carnegie-Mellon University.

Boyan, J. A., and Moore, A. W. (1997). Using Prediction to Improve Combinatorial Optimization Search. Proceedings of AI-STATS-97.

Bertsekas, D. P., and Tsitsiklis, J. N. (1996). *Neuro-Dynamic Programming.* Athena Scientific, Belmont, MA.

Bertsekas, D. P. (1997). Differential Training of Rollout Policies. In *Proc. of the 35th Allerton Conference on Communication, Control, and Computing.* Allerton Park, Ill.

Bertsekas, D. P., Tsitsiklis, J. N., and Wu, C. (1997). Rollout Algorithms for Combinatorial Optimization. *Journal of Heuristics.*

Brelaz, D. (1979). New Methods to Color Vertices of a Graph. *Communications of the ACM*, 22:251–256.

Healy, P., and Moll, R. (1995). A New Extension to Local Search Applied to the Dial-A-Ride Problem. *European Journal of Operations Research*, 8: 83–104.

Johnson, D., Aragon, C., McGeoch, L., and Schevon, C. (1991). Optimization by Simulated Annealing: An Experimental Evaluation; Part II, Graph Coloring and Number Partitioning. *Operations Research*, 39:378–406.

Kubo, M. and Kasugai, H. (1990). Heuristic Algorithms for the Single Vehicle Dial-A-Ride Problem. *Journal of Operations Research*, 33:354–364.

Papadimitriou, C. H., and Steiglitz, K. (1992). *Combinatorial Optimization: Algorithms and Complexity.* Prentice Hall, Englewood Cliffs, NJ.

Psaraftis, H. N. (1983). K-interchange Procedures for Local Search in a Precedence-Constrained Routing Problem. *European Journal of Operations Research*, 13:391–402.

Tesauro, G., and Galperin, G. R. (1996). On-line Policy Improvement using Monte-Carlo Search. In *Advances in Neural Information Processing: Proceedings of the Ninth Conference.* MIT Press.

Zhang, W. and Dietterich, T. G. (1995). A Reinforcement Learning Approach to Job-Shop Scheduling. In *Proceedings of the Fourteenth International Joint Conference on*